Working Paper Series
ISSN 1177-777X

COMPOSITIONAL SUPERVISOR SYNTHESIS WITH STATE MERGING AND TRANSITION REMOVAL

Sahar Mohajerani, Robi Malik, Martin Fabian

Working Paper: 02/2016
January 26, 2016

Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, 3240
New Zealand

COMPOSITIONAL SUPERVISOR SYNTHESIS WITH STATE MERGING AND TRANSITION REMOVAL

Sahar Mohajerani
Vehicle Dynamics and Active Safety Center
Volvo Cars Corporation
Göteborg, Sweden
sahar.mohajerani@volvocars.com

Robi Malik
Department of Computer Science
The University of Waikato
Hamilton, New Zealand
robi@waikato.ac.nz

Martin Fabian
Department of Signals and Systems
Chalmers University of Technology
Göteborg, Sweden
fabian@chalmers.se

January 26, 2016

Abstract

This working paper proposes a framework to obtain memory-efficient supervisors for large discrete event systems, which are least restrictive, controllable, and nonblocking. The approach combines compositional synthesis and state-based abstraction with transition removal to mitigate the state-space explosion problem and reduce the memory requirements. Hiding and nondeterminism after abstraction are also supported. To ensure least restrictiveness after transition removal, the synthesised supervisor has the form of cascaded maps representing the safe states. These maps have lower space complexity than previous automata-based supervisors. The algorithm has been implemented in the DES software tool Supremica and applied to compute supervisors for several large industrial models. The results show that supervisor maps can be computed efficiently and in many cases require less memory than automata-based supervisors.

1 Introduction

Supervisory control theory [25] provides a general framework to synthesise *supervisors* for discrete event systems, which restrict the system behaviour such that a given *specification* is fulfilled. To synthesise a supervisor, the set of states in which the specification is satisfied, the so-called *safe states*, is calculated. The straightforward approach to find the safe states

explores the complete system state space. This may be a problem for systems with a large number of components due to *state-space explosion*, which can cause the synthesis algorithm to take prohibitively long time or result in supervisors that are too large to implement.

To overcome state-space explosion, various approaches for modular, compositional, and symbolic synthesis have been proposed. With *modular* synthesis [2,4], the calculated supervisor is only least restrictive and controllable, but not necessarily nonblocking. *Compositional* methods [10] are based on the idea of repeatedly simplifying and computing partial supervisors while composing the system gradually. In [7, 27, 33], natural projection with additional restrictions of *output control consistency* or *local control consistency* is used for simplification. The methods in [9,15] allow for the removal of observable events through *hiding*. In [9], a least restrictive, controllable and nonblocking supervisor is constructed in symbolic form using *state labels*. Yet, the state labels are also needed during abstraction, which makes some abstraction impossible. State labels are avoided in [15], where a two-pass algorithm for compositional synthesis is proposed. Yet, the supervisor is an over-approximation and an additional nonblocking check is necessary.

In [21], *weak synthesis observation equivalence* is used to merge states, which has better abstraction potential compared to projection-based abstraction [22]. In addition to state merging, *halfway synthesis* [9] is used to remove blocking states early. To enable the supervisor to make correct control decisions, hiding is avoided and renaming is used to resolve nondeterminism after abstraction. The result is a set of automata that form a *modular* least restrictive, controllable, and nonblocking supervisor. Having modular supervisors reduces the memory requirement to some extent.

While state merging and state removal can reduce the complexity, there usually are a lot more transitions than states, and removing transitions can further reduce the memory usage. Transition removal is problematic in [21] where the supervisors have the form of automata and may fail to be least restrictive after the removal of transitions. A solution using *redirection maps* is described in [20]. However, it is not straightforward for the redirection maps to be combined with renaming and state merging abstraction, and therefore transition removal is the only abstraction method in [20].

Another approach is to compute *symbolic* representations of supervisors. In [14], Petri Net supervisors are represented in the form of *linear constraints*. In [30], the set of safe states of a system of synchronised automata is calculated in the form of a *binary decision diagram (BDD)* [5], which in [17] is attached to the original model to define a supervisor in the form of *guard formulas*. While these methods provide least restrictive, controllable and nonblocking supervisors, the resulting constraints and guard formulas may be complicated despite substantial effort to minimise them.

This working paper presents an improved compositional synthesis approach that produces more memory-efficient supervisors. Hiding and nondeterminism are supported. Moreover, the state-based abstraction methods [21] are enhanced with *certainly unsupervisable states* [31] and combined with transition removal [20].

This becomes possible by producing a supervisor in the form of a *cascaded map* that represents the set of safe states. This is similar to forming hierarchical groups of states and transitions as in Statecharts [11], except that the groupings are computed automatically as part of the abstraction process. The cascaded map is compact and used directly to control the system without the need to extract complicated guards. The space complexity of supervisor maps and supervisor automata [21] is compared, and it is shown that the supervisor maps require less memory. This is particularly important when the supervisor is implemented in devices with limited memory such as PLCs.

The improved compositional synthesis algorithm has been implemented in the DES software tool Supremica [1] and applied to compute supervisors for several large industrial models. The performance of the algorithm is compared with the previously implemented compositional synthesis algorithm [21]. Both algorithms successfully compute supervisors, even for systems with more than 10^{17} reachable states, within a few seconds or minutes. For most examples, the supervisor maps require less memory than the supervisor automata as is expected from the space complexity analysis.

In the following, Section 2 briefly introduces the background of supervisory control theory. Next, Section 3 explains the framework of compositional synthesis. Section 4 presents different ways of computing abstractions that preserve the synthesis result. Section 5 describes the control architecture based on cascaded maps, while Section 6 briefly describes the automata-based control architecture [21] and compares it to the map-based architecture. Then, Section 7 shows experimental results obtained by applying the algorithms to several benchmark examples, and Section 8 adds concluding remarks.

2 Preliminaries

2.1 Events and Languages

Discrete event systems [25] are modelled using events and states. *Events* represent incidents that cause transitions from one state to another and are taken from a finite alphabet Σ. For the purpose of supervisory control, the alphabet is partitioned into two disjoint subsets, the set Σ_c of *controllable* events and the set Σ_u of *uncontrollable* events. Controllable events can be disabled by a supervising agent, while uncontrollable events are always enabled. There are three special events called τ_c, τ_u, and ω. The *silent controllable* event $\tau_\mathrm{c} \in \Sigma_\mathrm{c}$ and the *silent uncontrollable* event $\tau_\mathrm{u} \in \Sigma_\mathrm{u}$ label transitions that are not taken by any component other than the one being considered. The *termination event* $\omega \in \Sigma_\mathrm{c}$ is used to show completion of a task.

The set of all finite *traces* of events from Σ, including the *empty trace* ε, is denoted by Σ^*. A subset $L \subseteq \Sigma^*$ is called a *language*. The *concatenation* of two traces $s, t \in \Sigma^*$ is written as st. The *natural projection* $P\colon \Sigma^* \to (\Sigma \setminus \{\tau_\mathrm{c}, \tau_\mathrm{u}\})^*$ is the operation that removes from traces $s \in \Sigma^*$ all occurrences of the silent events τ_c and τ_u.

2.2 Finite-State Automata

System behaviours are typically modelled by deterministic automata, but nondeterministic automata may arise as intermediate results during abstraction.

Definition 1 A (nondeterministic, finite) *automaton* is a tuple $G = \langle \Sigma, Q, Q^\circ, \to \rangle$, where Σ is a finite set of events, Q is a finite set of *states*, $Q^\circ \subseteq Q$ is the set of *initial states*, $\to \subseteq Q \times \Sigma \times Q$ is the *state transition relation*.

The transition relation is written in infix notation $x \xrightarrow{\sigma} y$, and is extended to traces and languages in the standard way. For example, $x \xrightarrow{\tau_u^* \sigma} y$ means that there exists a possibly empty sequence of τ_u-transitions followed by a σ-transition leading from state x to y. Furthermore, $x \xrightarrow{s}$ means $x \xrightarrow{s} y$ for some $y \in Q$. These notations also apply to state sets and to automata: $X \xrightarrow{s} Y$ for $X, Y \subseteq Q$ means $x \xrightarrow{s} y$ for some $x \in X$ and $y \in Y$, and $G \xrightarrow{s} x$ means $Q^\circ \xrightarrow{s} x$.

The termination event ω denotes completion of a task and does not appear anywhere else but to mark such completions. It is required that states reached by ω do not have any outgoing transitions, i.e., if $x \xrightarrow{\omega} y$ then there does not exist $\sigma \in \Sigma$ such that $y \xrightarrow{\sigma}$. This ensures that the termination event, if it occurs, is always the final event of any trace. The traditional set of marked states is $Q^\omega = \{\, x \in Q \mid x \xrightarrow{\omega} \,\}$ in this notation.

The interaction between automata is modelled by lock-step synchronisation [12]. An event shared between a set of automata, must be executed by all the automata together synchronously, while events used by only one automaton (and the silent events τ_c and τ_u) are executed by only that automaton.

Definition 2 The *synchronous composition* of two automata $G_1 = \langle \Sigma_1, Q_1, Q_1^\circ, \to_1 \rangle$ and $G_2 = \langle \Sigma_2, Q_2, Q_2^\circ, \to_2 \rangle$ is

$$G_1 \parallel G_2 = \langle \Sigma_1 \cup \Sigma_2, Q_1 \times Q_2, Q_1^\circ \times Q_2^\circ, \to \rangle \tag{1}$$

where

- $(x_1, x_2) \xrightarrow{\sigma} (y_1, y_2)$, if $\sigma \in (\Sigma_1 \cap \Sigma_2) \setminus \{\tau_c, \tau_u\}$, $x_1 \xrightarrow{\sigma}_1 y_1$, and $x_2 \xrightarrow{\sigma}_2 y_2$;
- $(x_1, x_2) \xrightarrow{\sigma} (y_1, x_2)$, if $\sigma \in (\Sigma_1 \setminus \Sigma_2) \cup \{\tau_c, \tau_u\}$ and $x_1 \xrightarrow{\sigma}_1 y_1$;
- $(x_1, x_2) \xrightarrow{\sigma} (x_1, y_2)$, if $\sigma \in (\Sigma_2 \setminus \Sigma_1) \cup \{\tau_c, \tau_u\}$ and $x_2 \xrightarrow{\sigma}_2 y_2$.

A common operation in compositional synthesis is *hiding*, which replaces events from a given set Υ by the silent events τ_c and τ_u, depending on whether the event to be replaced is controllable or uncontrollable.

Definition 3 [9] Let $G = \langle \Sigma, Q, Q^\circ, \to \rangle$ be an automaton and $\Upsilon \subseteq \Sigma$. The result of *controllability-preserving hiding* of Υ from G is $G \setminus_! \Upsilon = \langle \Sigma \setminus \Upsilon, Q, \to_!, Q^\circ \rangle$, where $\to_!$ is

obtained from $\to$ by replacing each transition $x \xrightarrow{\sigma} y$ such that $\sigma \in \Upsilon$ by $x \xrightarrow{\tau_\mathrm{c}} y$ if $\sigma \in \Sigma_\mathrm{c}$ or by $x \xrightarrow{\tau_\mathrm{u}} y$ if $\sigma \in \Sigma_\mathrm{u}$.

Another common automaton operation is the *quotient* modulo an equivalence relation on the state set.

Definition 4 Let X be a set. A relation $\sim \subseteq X \times X$ is called an *equivalence relation* on X if it is reflexive, symmetric, and transitive. Given an equivalence relation $\sim$ on X, the *equivalence class* of $x \in X$ is $[x] = \{\, x' \in X \mid x \sim x' \,\}$, and $X/{\sim} = \{\, [x] \mid x \in X \,\}$ is the set of all equivalence classes modulo $\sim$.

Definition 5 Let $G = \langle \Sigma, Q, Q^\circ, \to \rangle$ be an automaton and let $\sim \subseteq Q \times Q$ be an equivalence relation. The *quotient automaton* of G modulo $\sim$ is

$$G/{\sim} = \langle \Sigma, Q/{\sim}, \to/{\sim}, \tilde{Q}^\circ \rangle \ , \tag{2}$$

where $\to/{\sim} = \{\, ([x], \sigma, [y]) \mid x \xrightarrow{\sigma} y \,\}$ and $\tilde{Q}^\circ = \{\, [x^\circ] \mid x^\circ \in Q^\circ \,\}$.

2.3 Supervisory Control Theory

Supervisory control theory [25] provides a means to automatically compute a so-called *supervisor* that controls a given system to fulfil some desired functionality. Given an automata model of the possible behaviour of a physical system, called the *plant*, a supervisor is sought to restrict the behaviour in such a way that only a certain subset of the state space is reachable. The supervisor is implemented as a *control function*

$$\Phi \colon Q \to 2^{\Sigma \times Q} \tag{3}$$

that assigns to each state $x \in Q$ the set $\Phi(x)$ of transitions to be enabled in this state. That is, a transition $x \xrightarrow{\sigma} y$ with $\sigma \in \Sigma_\mathrm{c}$ will only be possible under the control of supervisor Φ if $(\sigma, y) \in \Phi(x)$. Transitions with uncontrollable events cannot be disabled, so it is required that $\Sigma_\mathrm{u} \times Q \subseteq \Phi(x)$ for all $x \in Q$.

In a deterministic system, the supervisor can be viewed more simply as a function

$$\Phi_\mathrm{det} \colon Q \to 2^{\Sigma_\mathrm{c}} \tag{4}$$

that assigns to each state the set of enabled controllable events. In a nondeterministic system, the supervisor (3) can be enable and disable transitions with controllable events individually, i.e., if a state x has more than one outgoing transition with the same controllable event σ, then the supervisor may disable some of them while leaving others enabled [9]. This is justified by the modelling assumptions of this working paper, where the nondeterminism is the result of abstraction, and it is assumed that the supervisor knows the complete system and can distinguish all transitions.

Definition 6 [9] $G_1 = \langle \Sigma, Q_1, Q_1^\circ, \rightarrow_1 \rangle$ is a *subautomaton* of $G_2 = \langle \Sigma, Q_2, Q_2^\circ, \rightarrow_2 \rangle$, written $G_1 \subseteq G_2$, if $Q_1 \subseteq Q_2$, $Q_1^\circ \subseteq Q_2^\circ$, and $\rightarrow_1 \subseteq \rightarrow_2$.

A subautomaton K of G consists of a subset of the states and transitions of G. It represents a supervisor that enables only those transitions present in K, i.e., it implements the control function

$$\Phi_K(x) = (\Sigma_\mathrm{u} \times Q) \cup \{\, (\sigma, y) \in \Sigma_\mathrm{c} \times Q \mid x \xrightarrow{\sigma}_K y \,\} \; . \tag{5}$$

Not every subautomaton of G can be implemented through control—the property of *controllability* [25] characterises those behaviours that can be implemented.

Definition 7 [9] Let $G = \langle \Sigma, Q_G, Q_G^\circ, \rightarrow_G \rangle$ and $K = \langle \Sigma, Q_K, Q_K^\circ, \rightarrow_K \rangle$ such that $K \subseteq G$. Then K is called *controllable* in G if, for all states $x \in Q_K$ and $y \in Q_G$ and for every uncontrollable event $\upsilon \in \Sigma_\mathrm{u}$ such that $x \xrightarrow{\upsilon}_G y$, it also holds that $x \xrightarrow{\upsilon}_K y$.

If a subautomaton K is controllable in G, then K must contain all uncontrollable transitions in G that originate from a state of K. Controllability ensures that the supervisor can be implemented without disabling any uncontrollable events. In addition to controllability, the supervised behaviour is typically required to be *nonblocking*.

Definition 8 [16] An automaton $G = \langle \Sigma, Q, Q^\circ, \rightarrow \rangle$ is *nonblocking*, if for every state $x \in Q$ and every trace $s \in (\Sigma \setminus \{\omega\})^*$ such that $G \xrightarrow{s} x$, there exists $t \in (\Sigma \setminus \{\omega\})^*$ such that $x \xrightarrow{t\omega}$.

Given a plant automaton G, the objective of *supervisor synthesis* [25] is to compute a subautomaton $K \subseteq G$, which is controllable and nonblocking and restricts the behaviour of G as little as possible. The set of subautomata of G forms a lattice, and the upper bound of a set of controllable and nonblocking subautomata in this lattice is again controllable and nonblocking.

Theorem 1 [9] Let $G = \langle \Sigma, Q, Q^\circ, \rightarrow \rangle$ be an automaton. There exists a unique subautomaton $\sup\mathcal{C}(G) \subseteq G$ such that $\sup\mathcal{C}(G)$ is nonblocking and controllable in G, and such that for every subautomaton $S \subseteq G$ that is also nonblocking and controllable in G, it holds that $S \subseteq \sup\mathcal{C}(G)$.

The subautomaton $\sup\mathcal{C}(G)$ represents the unique *least restrictive* controllable and nonblocking sub-behaviour of G that can be achieved by any possible supervisor. It can be computed by iteratively removing blocking states and states leading to blocking states via uncontrollable events, according to the following definitions, until a fixpoint is reached [9].

Definition 9 [9] The *restriction* of $G = \langle \Sigma, Q, Q^\circ, \rightarrow \rangle$ to $X \subseteq Q$ is

$$G_{|X} = \langle \Sigma, Q, \rightarrow_{|X}, Q^\circ \cap X \rangle \; , \tag{6}$$

where $\rightarrow_{|X} = \{\, (x, \sigma, y) \in \rightarrow \mid x, y \in X \,\} \cup \{\, (x, \omega, y) \in \rightarrow \mid x \in X \,\}$.

Definition 10 [9] Let $G = \langle \Sigma, Q, Q^\circ, \to \rangle$ be an automaton. The *synthesis step* operator $\Theta_G \colon 2^Q \to 2^Q$ for G is defined as $\Theta_G(X) = \Theta_G^{\mathrm{cont}}(X) \cap \Theta_G^{\mathrm{nonb}}(X)$, where

$$\Theta_G^{\mathrm{cont}}(X) = \{\, x \in Q \mid \text{for all transitions } x \xrightarrow{\upsilon} y \text{ with } \upsilon \in \Sigma_{\mathrm{u}} \text{ it holds that } y \in X \,\} \; ; \tag{7}$$

$$\Theta_G^{\mathrm{nonb}}(X) = \{\, x \in Q \mid x \xrightarrow{t\omega}_{|X} \text{ for some } t \in \Sigma^* \,\} \; . \tag{8}$$

Given a state set $X \subseteq Q$, the operator Θ_G^{cont} removes from X any states that have an uncontrollable successor not contained in X, and Θ_G^{nonb} removes any states from where it is not possible to execute the termination event ω using only transitions contained in X. Thus, Θ_G^{cont} captures controllability and Θ_G^{nonb} captures the nonblocking property. Both operators and their combination Θ_G are monotonic, and it follows by the Knaster-Tarski theorem [29] that they have greatest fixpoints. The least restrictive synthesis result $\sup\mathcal{C}(G)$ is obtained by restricting G to the greatest fixpoint of Θ_G.

Theorem 2 [9] Let $G = \langle \Sigma, Q, Q^\circ, \to \rangle$. The synthesis step operator Θ_G for G has a greatest fixpoint $\hat{\Theta}_G = \mathrm{gfp}\Theta_G \subseteq Q$, such that $G_{|\hat{\Theta}_G}$ is the greatest subautomaton of G that is both controllable in G and nonblocking, i.e., $\sup\mathcal{C}(G) = G_{|\hat{\Theta}_G}$.

The operator $\sup\mathcal{C}$ only defines the synthesis result for a plant automaton G. In order to apply this synthesis to control problems that also involve *specifications*, the transformation proposed in [9] is used. Specification automata are transformed into plants by adding, for every uncontrollable event that is not enabled in a state, a transition to a new blocking state $\bot$. This transforms all potential controllability problems into potential blocking problems.

3 Compositional Synthesis

This section gives an overview of the compositional synthesis framework. The input to compositional synthesis is an arbitrary set of deterministic automata representing the plant to be controlled,

$$\mathcal{G}_0 = \{G_1, G_2, \ldots, G_n\} \; . \tag{9}$$

The objective is to calculate a supervisor that constrains the behaviour of $\mathcal{G}_0$ to its least restrictive nonblocking sub-behaviour, by disabling only controllable events.

Compositional synthesis works by repeated abstraction of system components. Using abstraction, some components G_i in (9) are replaced by simpler versions G_i'. If this is no longer possible, then some components are composed, i.e., replaced by their synchronous composition, which can then be simplified using abstraction. This procedure eventually leads to a single automaton that after abstraction is simpler than the original system. The final step is to perform standard synthesis [25] on this last relatively small automaton.

To obtain a supervisor from the synthesis result, Theorem 2 suggests that it is enough to compute the set $\hat{\Theta}_{\mathcal{G}_0}$ of safe states. This set can be obtained from the synthesis result of the

final abstraction, if it is known how the states of the original system $\mathcal{G}_0$ have been abstracted. Therefore compositional synthesis does not only keep track of an abstracted system $\mathcal{G}$, but also of a map μ that links the states of the original plant $\mathcal{G}_0$ at the start of the algorithm to the states of the current abstraction. The abstracted system $\mathcal{G}$ and the map μ are combined in a *synthesis pair*, which is the main data structure manipulated by the compositional synthesis algorithm.

Definition 11 Let $\mathcal{G}_0$ be a set of automata. A *synthesis pair* for $\mathcal{G}_0$ is a pair $(\mathcal{G};\mu)$, where $\mathcal{G}$ is a set of automata and μ is a map $\mu\colon Q_{\mathcal{G}_0} \to Q_{\mathcal{G}}$.

In Definition 11, $Q_{\mathcal{G}}$ represents the set of states of the synchronous composition of $\mathcal{G}$. Synthesis pairs replace *synthesis triples* [21]. Synthesis pairs are simpler than synthesis triples, which in addition contain supervisor automata and renaming information, while synthesis pairs only contain the information that leads to the set of safe states.

While manipulating synthesis pairs, the compositional synthesis algorithm maintains the invariant that all generated pairs have the same synthesis result, which is equivalent to the least restrictive solution of the original control problem (9). Every abstraction step must ensure that the synthesis result is the same as it would have been for the non-abstracted components. This property is called *synthesis equivalence*.

Definition 12 Let $(\mathcal{G}_1;\mu_1)$ and $(\mathcal{G}_2;\mu_2)$ be two synthesis pairs for $\mathcal{G}_0$. Then $(\mathcal{G}_1;\mu_1)$ and $(\mathcal{G}_2;\mu_2)$ are *synthesis equivalent*, written $(\mathcal{G}_1;\mu_1) \simeq_{\text{synth}} (\mathcal{G}_2;\mu_2)$, if for all $x \in Q_{\mathcal{G}_0}$ it holds that $\mu_1(x) \in \hat{\Theta}_{\mathcal{G}_1}$ if and only if $\mu_2(x) \in \hat{\Theta}_{\mathcal{G}_2}$.

At the beginning of compositional synthesis on input $\mathcal{G}_0$, no states have been abstracted, so the initial synthesis pair is $(\mathcal{G}_0;\mathrm{id})$, where $\mathrm{id}\colon Q_{\mathcal{G}_0} \to Q_{\mathcal{G}_0}$ is the identity map, i.e, $\mathrm{id}(x) = x$ for all $x \in Q_{\mathcal{G}_0}$. This initial pair is abstracted repeatedly such that synthesis equivalence is preserved,

$$(\mathcal{G}_0;\mathrm{id}) \simeq_{\text{synth}} (\mathcal{G}_1;\mu_1) \simeq_{\text{synth}} \cdots \simeq_{\text{synth}} (\mathcal{G}_k;\mu_k)\ . \tag{10}$$

Some of these steps replace an automaton in $\mathcal{G}_k$ by an abstraction, others reduce the number of automata in $\mathcal{G}_k$ by synchronous composition. The algorithm terminates when $\mathcal{G}_k$ contains a single automaton, for which the synthesis solution $\hat{\Theta}_{\mathcal{G}_k}$ is computed. As each step in (10) preserves synthesis equivalence, a state x of the original system $\mathcal{G}_0$ is safe, i.e., $x \in \hat{\Theta}_{\mathcal{G}_0}$, if and only if the state that x is mapped to by the final map μ_k is contained in the final synthesis result, i.e., $\mu_k(x) \in \hat{\Theta}_{\mathcal{G}_k}$. This information is enough to implement a supervisor as explained below in Section 5.

4 Abstraction Methods

Compositional synthesis is a series of steps to rewrite synthesis pairs into equivalent synthesis pairs such that synthesis equivalence is preserved. In the following, Section 4.1 introduces

the general concepts of hiding and state-wise synthesis abstraction. Then Sections 4.2–4.5 describe the methods to simplify and compose automata, namely the removal of *certainly unsupervisable states* [31], the merging of states based on *synthesis observation equivalence* [21], *transition removal* [20], and the fallback method of *synchronous composition*. All the methods are adapted in the synthesis pair framework, and it is shown how the supervisor map is constructed after each abstraction. Finally, Section 4.6 gives an example to show how these methods work together to compute a supervisor.

4.1 Hiding and State-wise Synthesis Abstraction

Simplification in compositional synthesis is based on *local* events. An event that appears in only one of the components of the system (9) is called local. These events are not needed for synchronisation, so their identity is irrelevant and they can be removed using controllability-preserving hiding (Definition 3). This replaces them by the silent events τ_{c} and τ_{u}, which are used by most simplification methods.

After hiding, various methods have been proposed to simplify a single automaton without further considering the remainder of the system [9, 20, 21]. The following definition of state-wise synthesis abstraction provides a general description that captures all the abstractions in the following subsections.

Definition 13 Let $G = \langle \Sigma, Q_G, Q_G^\circ, \to_G \rangle$ and $H = \langle \Sigma, Q_H, Q_H^\circ, \to_H \rangle$ be two automata. A map $\mu\colon Q_G \to Q_H$ is a *state-wise synthesis abstraction* from G to H, if for all automata $T = \langle \Sigma_T, Q_T, Q_T^\circ, \to_T \rangle$ and for all states $x_T \in Q_T$ and $x_G \in Q_G$ the following equivalence holds:

$$(x_G, x_T) \in \hat{\Theta}_{G\|T} \text{ if and only if } (\mu(x_G), x_T) \in \hat{\Theta}_{H\|T} \ .$$

State-wise synthesis abstraction is defined by a map that relates states of a single automaton G before and after abstraction if they have the same synthesis behaviour in every possible environment T. That is, no matter what other automaton G is composed with, either both the original and the abstracted states are removed in synthesis, or none of them are removed. The following proposition confirms that the state-wise synthesis abstraction of an automaton preserves synthesis equivalence of the complete synthesis pairs.

Proposition 3 Let $(\mathcal{G}; \mu)$ be a synthesis pair for $\mathcal{G}_0$ with $\mathcal{G} = \{G_1, \ldots, G_n\}$. Let μ_1 be a state-wise synthesis abstraction from G_1 to H_1, let $\mathcal{H} = \{H_1, G_2, \ldots, G_n\}$, and let $\tilde{\mu}_1\colon Q_{\mathcal{G}} \to Q_{\mathcal{H}}$ such that $\tilde{\mu}_1(y_1, y_2, \ldots, y_n) = (\mu_1(y_1), y_2, \ldots, y_n)$. Then $(\mathcal{G}; \mu) \simeq_{\text{synth}} (\mathcal{H}; \tilde{\mu}_1 \circ \mu)$.

Proof. Let $x \in Q_{\mathcal{G}_0}$ and $\mu(x) = (y_1, \ldots, y_n)$.

First assume $\mu(x) \in \hat{\Theta}_{\mathcal{G}}$. This means $(y_1, \ldots, y_n) \in \hat{\Theta}_{\mathcal{G}} = \hat{\Theta}_{G_1\|\cdots\|G_n}$. Considering $T = G_2\|\cdots\|G_n$ in Definition 13, it holds that $\tilde{\mu}_1(\mu(x)) = \tilde{\mu}_1(y_1, y_2, \ldots, y_n) = (\mu_1(y_1), y_2, \ldots, y_n) \in \hat{\Theta}_{H_1\|G_2\|\cdots\|G_n} = \hat{\Theta}_{\mathcal{H}}$.

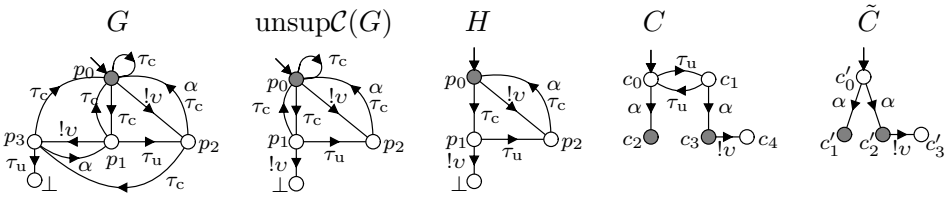

Figure 1: Example automata and abstractions.

Conversely assume that $\tilde{\mu}_1(\mu(x)) \in \hat{\Theta}_{\mathcal{H}}$. This means $(\mu_1(y_1), y_2, \ldots, y_n) = \tilde{\mu}_1(y_1, \ldots, y_n) = \tilde{\mu}_1(\mu(x)) \in \hat{\Theta}_{\mathcal{H}} = \hat{\Theta}_{H_1 \| G_2 \| \cdots \| G_n}$. Considering $T = G_2 \| \cdots \| G_n$ in Definition 13, it holds that $\mu(x) = (y_1, \ldots, y_n) \in \hat{\Theta}_{G_1 \| \cdots \| G_n} = \hat{\Theta}_{\mathcal{G}}$. □

4.2 Removal of Certainly Unsupervisable States

For some states of an automaton G, it is known that they must be avoided by synthesis in every possible context. That is, no matter what other automata are later composed with G, it is clear that these states are unsafe. Blocking states are examples of such states, but there are more states with this property. *Halfway synthesis* [9] is a simple method to find and remove such states. Even though halfway synthesis works well in compositional synthesis [21], it does not identify all the removable states. The largest set of removable states is known as the set of certainly unsupervisable states [31]. In the following, their removal is adapted to the synthesis pair framework.

Definition 14 [31] Let $G = \langle \Sigma, Q, Q^\circ, \rightarrow \rangle$ be an automaton. The *certainly unsupervisable state set* of G is

$$U(G) = \{\, x \in Q \mid \text{for every automaton } T = \langle \Sigma, Q_T, Q_T^\circ, \rightarrow_T \rangle \text{ and every state } x^T \in Q_T \text{ it holds that } (x, x^T) \notin \hat{\Theta}_{G \| T} \,\} \; . \quad (11)$$

A state x of G is certainly unsupervisable, if there exists no other automaton T such that the state x is present in the least restrictive synthesis result $\hat{\Theta}_{G \| T}$. If a state is certainly unsupervisable, it is known that this state will be removed by every synthesis. If such states are encountered in an automaton during compositional synthesis, they can be removed before composing this automaton further.

Example 1 Consider automaton G in Figure 1. Event α is controllable, and $!\upsilon$ is uncontrollable. Clearly the blocking state $\perp$ is certainly unsupervisable. In addition, state p_3 is also certainly unsupervisable, because of the silent uncontrollable transition $p_3 \xrightarrow{\tau_u} \perp$. As this transition is silent, no other component disables it, and as it is uncontrollable, the supervisor

cannot disable it. Therefore, if the automaton ever enters state p_3, blocking is unavoidable. State p_1, however, may still be safe as some other components may disable event $!\upsilon$.

A cubic-complexity algorithm [31] can be used to find all the certainly unsupervisable states in an automaton. Then the automaton can be simplified according to the following definition.

Definition 15 [31] Let $G = \langle \Sigma, Q, Q^\circ, \to \rangle$ be an automaton. The result of *unsupervisability removal* from G is the automaton

$$\mathrm{unsup}\mathcal{C}(G) = \langle \Sigma, Q_{\mathrm{unsup}}, Q^\circ, \to_{\mathrm{unsup}} \rangle \ , \tag{12}$$

where

$$Q_{\mathrm{unsup}} = (Q \setminus U(G)) \cup \{\bot\} \ ; \tag{13}$$

$$\begin{aligned} \to_{\mathrm{unsup}} = {} & \{\, (x,\sigma,y) \in \to \mid x, y \notin U(G) \,\} \cup \\ & \{\, (x,\upsilon,\bot) \in \to \mid x \notin U(G),\, \upsilon \in \Sigma_{\mathrm{u}} \cup \{\omega\}, \text{ and } x \xrightarrow{\upsilon} U(G) \,\} \ . \end{aligned} \tag{14}$$

Here and in the following, the state $\bot$ is assumed to be a state without any outgoing transitions. The automaton resulting from unsupervisability removal has a state set, which is obtained by removing certainly unsupervisable states except for $\bot$. All controllable transitions to certainly unsupervisable states are removed, as these transitions can always be disabled by the supervisor and therefore never appear in the final synthesis result. Uncontrollable and ω-transitions to certainly unsupervisable states, however, are retained because they are needed to inform future synthesis steps.

Example 2 Consider again automaton G in Figure 1. To obtain $\mathrm{unsup}\mathcal{C}(G)$, the certainly unsupervisable states p_3 and $\bot$, identified in Example 1, are replaced by the single blocking state $\bot$. The resulting automaton is shown in Figure 1.

Proposition 4 Let $G = \langle \Sigma, Q, Q^\circ, \to \rangle$ be an automaton and let the mapping $\mu \colon Q \to Q_{\mathrm{unsup}}$ be defined by:

$$\mu(x) = \begin{cases} x, & \text{if } x \in Q \setminus U(G); \\ \bot, & \text{if } x \in U(G). \end{cases} \tag{15}$$

Then μ is a state-wise synthesis abstraction from G to $\mathrm{unsup}\mathcal{C}(G)$.

After removing the unsupervisable states, a map is generated that maps the unsupervisable states to $\bot$. Proposition 4 confirms that this is a state-wise synthesis abstraction map. The proof follows from the results of [31].

4.3 Abstraction by State Merging

In addition to the removal of certainly unsupervisable states, it is of interest to merge states with equivalent behaviour. While no method is known to identify all equivalent states in compositional synthesis, several approximations have been proposed [21]. One of these is weak synthesis observation equivalence, defined as follows.

Definition 16 [21] Let $G = \langle \Sigma, Q, Q^\circ, \to \rangle$ be an automaton. An equivalence relation $\sim \subseteq Q \times Q$ is a *weak synthesis observation equivalence* on G if the following conditions hold for all $x_1, x_2 \in Q$ such that $x_1 \sim x_2$:

(i) If $x_1 \xrightarrow{\sigma} y_1$ for $\sigma \in \Sigma_\mathrm{c}$, then there exists a path $x_2 = x_2^0 \xrightarrow{\tau_1} \cdots \xrightarrow{\tau_n} x_2^n \xrightarrow{P(\sigma)} y_2^{n+1} \xrightarrow{\tau_{n+1}} \cdots \xrightarrow{\tau_m} y_2^{m+1} = y_2$ such that $y_1 \sim y_2$ and,

  a) if $\tau_i = \tau_\mathrm{c}$ for some $i \leq n$, then $x_1 \sim x_2^i$;

  b) if $y_2^i \xrightarrow{\tau_\mathrm{u}^*} z$ then $z \sim y_2^j$ for some $n+1 \leq j \leq m+1$;

  c) if $y_2^i \xrightarrow{\tau_\mathrm{u}^* \upsilon \tau_\mathrm{u}^*} z$ for some $\upsilon \in \Sigma_\mathrm{u} \setminus \{\tau_\mathrm{u}\}$, then $y_2 \xrightarrow{\tau_\mathrm{u}^* \upsilon \tau_\mathrm{u}^*} z'$ for some $z' \sim z$.

(ii) If $x_1 \xrightarrow{\upsilon} y_1$ for $\upsilon \in \Sigma_\mathrm{u}$, then there exist $t_2, u_2 \in \tau_\mathrm{u}^*$ such that $x_2 \xrightarrow{t_2 P(\upsilon) u_2} y_2$ and $y_1 \sim y_2$.

Proposition 5 Let $G = \langle \Sigma, Q, Q^\circ, \to \rangle$ be an automaton and let $\sim \subseteq Q \times Q$ be a weak synthesis observation equivalence on G. Let the mapping $\mu \colon Q \to Q/\!\sim$ be such that for all $x \in Q$ it holds that $\mu(x) = [x]$. Then μ is a state-wise synthesis abstraction from G to $G/\!\sim$.

After merging weakly synthesis observation equivalent states, each state x is mapped to the equivalence class $[x]$ in the abstracted automaton. Proposition 5 confirms that this is a state-wise synthesis abstraction map. The proof can be found in [19].

Example 3 Consider automaton C in Figure 1, with α controllable and $!\upsilon$ uncontrollable. An equivalence relation with $c_0 \sim c_1$ is a synthesis observation equivalence on C. Merging the equivalent states results in the synthesis equivalent nondeterministic automaton $\tilde{C} = C/\!\sim$ shown in Figure 1. Accordingly, the map μ_1 such that $\mu_1(c_0) = \mu_1(c_1) = c_0'$, $\mu_1(c_2) = c_1'$, $\mu_1(c_3) = c_2'$, and $\mu_1(c_4) = c_3'$ is a state-wise synthesis abstraction from C to $\tilde{C}$.

4.4 Synthesis Transition Removal

Another way of abstracting an automaton is by removing redundant transitions. Methods to remove transitions in compositional synthesis are described in [20], where *redirection maps* are used to enable the supervisor to make control decision for the removed transitions. These abstraction methods are not used in the compositional synthesis framework [21] as in it is not straightforward to combine redirection maps with renaming and merging states. However,

there usually are much more transitions than states in an automaton, and removing transitions can significantly decrease memory usage. All the transition removal methods in [20] are combined in the following general definitions, and it is shown that the need for redirection maps and the associated problems do not arise when synthesis pairs are used.

Definition 17 [20] Let $G = \langle \Sigma, Q, Q^\circ, \rightarrow \rangle$ be an automaton. A path

$$x_0 \xrightarrow{\tau_1} x_1 \xrightarrow{\tau_2} \cdots \xrightarrow{\tau_k} x_k \tag{16}$$

where $\tau_i \in \{\tau_\mathrm{c}, \tau_\mathrm{u}\}$ for $1 \leq i \leq k$, is a *weakly controllable path* if for all uncontrollable transitions $x_l \xrightarrow{\upsilon} y$ with $0 \leq l < k$ it holds that $\upsilon = \tau_\mathrm{u}$ and $y = x_j$ for some $0 \leq j \leq k$.

A weakly controllable path consists of only silent τ_u- and τ_c-transitions, and furthermore all uncontrollable transitions enabled along this path must use the silent uncontrollable event τ_u and lead to states along the path. By imposing this condition on the sequence of silent events after a controllable event, *synthesis transition removal* can be defined that preserves state-wise synthesis abstraction.

Definition 18 Let $G = \langle \Sigma, Q, Q^\circ, \rightarrow_G \rangle$ and $H = \langle \Sigma, Q, Q^\circ, \rightarrow_H \rangle$ be two automata and $\rightarrow_H \subseteq \rightarrow_G$. Automaton H is a result of *synthesis transition removal* from G if the following conditions hold for all transitions $x \xrightarrow{\sigma}_G y$.

(i) If $\sigma \in \Sigma_\mathrm{u}$ then $x \xrightarrow{\tau_\mathrm{u}^* P(\sigma) \tau_\mathrm{u}^*}_H y$.

(ii) If $\sigma \in \Sigma_\mathrm{c}$ then there exist $t \in \{\tau_\mathrm{u}\}^*$ and $u \in \{\tau_\mathrm{c}, \tau_\mathrm{u}\}^*$ such that $x \xrightarrow{tP(\sigma)}_H z \xrightarrow{u}_H y$, and $z \xrightarrow{u}_G y$ is a weakly controllable path.

Example 4 Consider automaton unsup$\mathcal{C}(G)$ in Figure 1 where α is a controllable event and $!\upsilon$ is an uncontrollable event. Applying synthesis transition removal to unsup$\mathcal{C}(G)$ removes transitions $p_1 \xrightarrow{\tau_\mathrm{c}} p_0$ because of $p_1 \xrightarrow{\tau_\mathrm{u}} p_2 \xrightarrow{\tau_\mathrm{c}} p_0$, and $p_0 \xrightarrow{\tau_\mathrm{c}} p_0$ because of $p_0 \xrightarrow{\varepsilon} p_0$. Their removal results in H shown in Figure 1.

No states are merged when removing transitions, so the identity map is a state-wise synthesis abstraction map after synthesis transition removal.

Proposition 6 Let H be the result of synthesis transition removal from G. Then the identity map id is a state-wise synthesis abstraction from G to H.

A proof that transition removal preserves state-wise synthesis abstraction is given in [18].

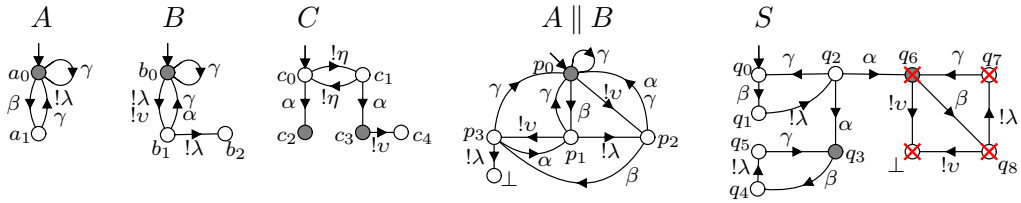

Figure 2: Example of compositional synthesis.

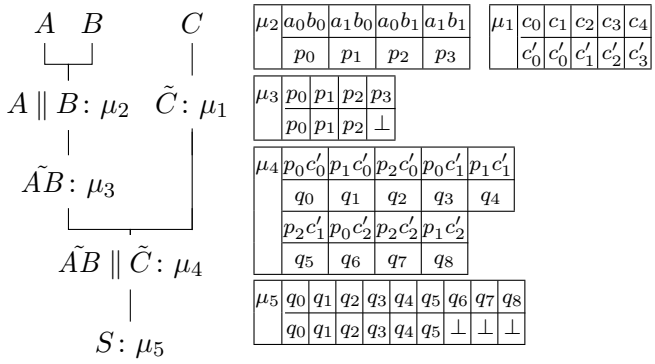

Figure 3: Supervisor maps for Figure 2.

4.5 Synchronous Composition

If no abstraction is possible, then compositional synthesis needs to compose some automata. It is always possible to compose two automata in the set of uncontrolled plants without affecting the synthesis result. This basic method is included here for the sake of completeness. It does not contribute to simplification on its own, but usually the result of composition can be simplified again afterwards.

Proposition 7 Let $(\mathcal{G};\mu)$ be a synthesis pair for $\mathcal{G}_0$ with $\mathcal{G} = \{G_1,\ldots,G_n\}$, let $\mathcal{H} = \{G_1 \| G_2, G_3,\ldots,G_n\}$, and let $\mu_{12}\colon Q_\mathcal{G} \to Q_\mathcal{H}$ such that $\mu_{12}(y_1,\ldots,y_n) = ((y_1,y_2),y_3,\ldots,y_n)$. Then $(\mathcal{G};\mu) \simeq_{\text{synth}} (\mathcal{H};\mu_{12}\circ\mu)$.

4.6 Worked Example

This section gives an example of the complete process of compositional synthesis using the above propositions. Consider system $\mathcal{G}_0 = \{A, B, C\}$ shown in Figure 2. Events α, β, and γ are controllable, and $!\eta$, $!\lambda$ and $!\upsilon$ are uncontrollable. Compositional synthesis starts from

the initial synthesis pair:

$$(\{A, B, C\}; \mathrm{id}) \tag{17}$$

From here, the algorithm first hides event $!\eta$ from automaton C, as it is a local event, and simplifies C to the nondeterministic automaton $\tilde{C}$ as explained in Example 3. The states of C are linked to the states of $\tilde{C}$ by the state-wise synthesis abstraction map μ_1 shown in Figure 3. This map is extended to a map $\tilde{\mu}_1$ that links states of $A \parallel B \parallel C$ to states of $A \parallel B \parallel \tilde{C}$, as defined in Proposition 3. By Propositions 3 and 5, the initial synthesis pair (17) is synthesis equivalent to the following pair:

$$(\{A, B, \tilde{C}\}; \tilde{\mu}_1) \tag{18}$$

Next, automata A and B are composed as shown in Figure 2, and map μ_2, shown in Figure 3, is constructed to map the states of A and B to the states of $A \parallel B$. Its extension $\tilde{\mu}_2$ according to Proposition 7 leads to the next synthesis equivalent pair:

$$(\{A \parallel B, \tilde{C}\}; \tilde{\mu}_2 \circ \tilde{\mu}_1) \tag{19}$$

After the composition of A and B, the controllable events α and γ and the uncontrollable event $!\lambda$ become local and can be hidden, resulting in automaton $(A \parallel B) \setminus_! \{\gamma, \alpha, !\lambda\}$, shown as G in Figure 1. Automaton G is abstracted to automaton H in Figure 1 by removing certainly unsupervisable states and by synthesis transition removal as explained in Examples 1 and 4. This results in the abstracted automaton $\tilde{AB} = H$ and a state-wise synthesis abstraction map μ_3 that maps state p_3 of $A \parallel B$ to $\perp$. By Propositions 3, 4, and 6, its extension $\tilde{\mu}_3$ gives the next synthesis equivalent pair:

$$(\{\tilde{AB}, \tilde{C}\}; \tilde{\mu}_3 \circ \tilde{\mu}_2 \circ \tilde{\mu}_1) \tag{20}$$

The next step of compositional synthesis is to compose $\tilde{AB}$ and $\tilde{C}$, resulting in the automaton S in Figure 2, map μ_4 in Figure 3, and the final synthesis equivalent pair:

$$(\{\tilde{AB} \parallel \tilde{C}\}; \mu_4 \circ \tilde{\mu}_3 \circ \tilde{\mu}_2 \circ \tilde{\mu}_1) \tag{21}$$

Finally, a supervisor is calculated for $S = \tilde{AB} \parallel \tilde{C}$. This supervisor is obtained by removing the crossed out states from S in Figure 2, and it is presented as a final map μ_5 in Figure 3, which maps these removed states to $\perp$. As (17) and (21) are synthesis equivalent pairs, this final supervisor results in the same least restrictive behaviour as a supervisor synthesised monolithically for the original system $\mathcal{G}_0$.

5 State Representation Architecture

When compositional synthesis completes, a supervisor can be implemented using the constructed maps. At each step of compositional synthesis, a state-wise synthesis abstraction

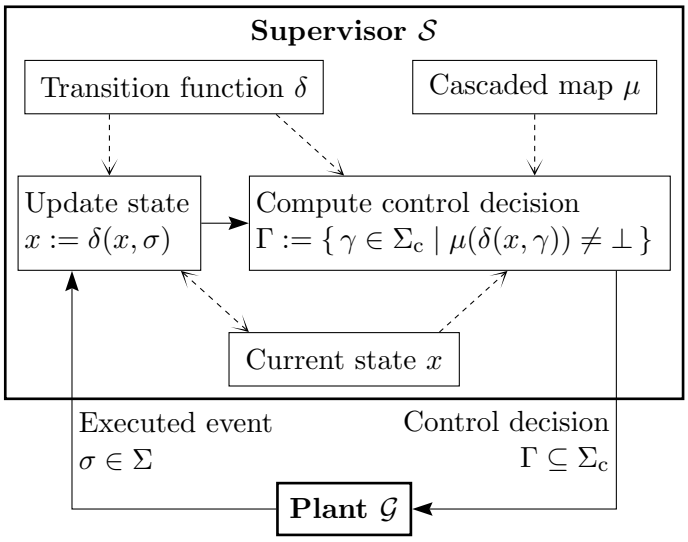

Figure 4: Control architecture.

map is constructed that relates the states of the abstracted system to the states in the previous step. This correspondence is propagated in such a way that the combination of all maps represents all the states that are reached under a least restrictive controllable and nonblocking supervisor.

The supervisor interacts with the plant following the control architecture in Figure 4. Initially, the supervisor assumes the initial state of the plant model as its current state x, and based on that issues an initial control decision $\Gamma \subseteq \Sigma_c$, containing the set of controllable events to be enabled. When the plant executes the next event $\sigma \in \Sigma$, the supervisor first updates its state x using the transition function $\delta\colon Q \times \Sigma \to Q$. This deterministic version of the transition relation $\to$ exists, as it is based on the original plant model before abstraction, which is deterministic. Then the new control decision is calculated by finding, for each controllable event $\gamma \in \Sigma_c$ the successor state $y = \delta(x, \gamma)$ and, if it is defined, checking the supervisor map to see whether it is a safe state. If the successor state y is defined and not mapped to the unsafe state $\bot$, then the controllable event γ is enabled by including it in the control decision Γ, otherwise it is disabled.

To show how the supervisor map works in detail, consider again the example in Figure 2 with maps in Figure 3. Assume the system $\mathcal{G}_0 = \{A, B, C\}$ is in the global state $a_0b_1c_0$. Inspection of the automata model shows that events α, β, and $!\eta$ are enabled in this state. Event α would lead the system to $a_0b_0c_2$. To look up this state, first the state pair a_0b_0 is combined to p_0 according to map μ_2, and p_0 is mapped to p_0 by map μ_3. Moreover, state c_2 is mapped to c_1' by map μ_1. Next, the combination p_0c_1' is mapped to q_3 by μ_4, which the

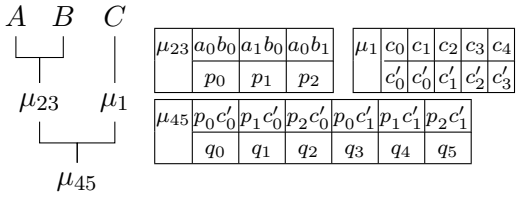

Figure 5: Compacted supervisor maps for Figure 2.

final map μ_5 maps to q_3. Thus the system state $a_0b_0c_2$ has an image in the final map, so $a_0b_0c_2$ is a safe state and α should be enabled. On the other hand, event β would lead the system to $a_1b_1c_0$. The state pair a_1b_1 is mapped to p_3 by μ_2, which is mapped to $\perp$ by μ_3. At this point, it becomes obvious that $a_1b_1c_0$ is a bad state. Therefore the supervisor disables β. This results in the control decision $\Gamma = \{\alpha\}$, which means to enable α and disable β. Event $!\eta$ is uncontrollable and cannot be disabled by the supervisor. This event leads to state $a_0b_1c_1$, which is mapped to q_2. This confirms that the supervisor is controllable.

The cascaded map representing the final supervisor can be further simplified by composing each synchronisation map and the abstraction map following it. For example, the composition of the synchronisation map μ_2 and the abstraction map μ_3 in Figure 3, is done by feeding the output of μ_2 directly into μ_3. This results in replacing μ_2 and μ_3 with $\mu_{23} = \mu_3 \circ \mu_2$, and likewise μ_4 and μ_5 can be replaced with $\mu_{45} = \mu_5 \circ \mu_4$. Moreover, the blocking state $\perp$ can be removed from all maps, if the absence of an entry for a given state is interpreted as that state being unsafe. The simplified maps are shown in Figure 5.

Complexity. The space complexity of the state representation supervisor is determined by the number of maps and their size. At the beginning of compositional synthesis, each automaton can be abstracted once, and afterwards there can be an abstraction step each time after automata have been composed. If there are n automata in the model, this gives n abstraction steps initially, plus $n-1$ abstraction steps after synchronous composition, as the number of components decreases by at least one after each synchronous composition. In the worst-case, each abstraction step results in one supervisor map, giving up to $2n-1$ maps in total. The size of these maps is determined by the number of states of the automata encountered in each step. If the largest automaton has $|Q|$ states, the worst-case space complexity to store the supervisor maps is $O((2n-1)|Q|) = O(n|Q|)$. Thus, the worst-case memory usage to store the state representation supervisor is linear in the number of components and in the number of states.

6 Automata-based Supervisor

Previous work [21] follows a similar compositional approach as described above to produce a supervisor in the form of automata instead of maps. The supervisor automata interact in synchronous composition with the system to track its states, and the supervisor disables any controllable events that are disabled by the supervisor automata. For this approach to work, care must be taken to ensure that the supervisor automata can be implemented.

In the example in Figure 2, two supervisor components are needed to disable the controllable transitions $p_2 \xrightarrow{\beta} p_3$ in $A \parallel B$ after removing the certainly unsupervisable states and $q_2 \xrightarrow{\alpha} q_6$ is S. However, the automaton S cannot be implemented as a supervisor because of the nondeterminism in state q_2, where it is not clear whether event α should be enabled or disabled. In [21], *renaming* is used to avoid nondeterminism after abstraction, and event α is replaced by two new events α_1 and α_2. The decision which of these events is enabled is made by a so-called *distinguisher*, which is derived from automaton C. This gives the supervisor automata shown in Figure 6, where RC is the distinguisher, and $\tilde{AB}$ and RS are the renamed supervisor components.

Another issue arises when S, shown in Figure 2, is used as a supervisor automaton as it does not include the removed transition $q_1 \xrightarrow{\gamma} q_0$. Then the supervisor disables the controllable event γ in the global state $a_1b_1c_0$ and thus is not a least restrictive supervisor. Therefore, in [21] transition removal is not used as an abstraction method, so this transition is present in RS. Transition removal is used in [20], where a so-called *redirection map* is proposed, which maps the removed transition to the alternative path. Yet, the redirection map cannot be represented in the form of automata, and it is not straightforward to combine redirection maps with other abstraction methods or with renaming.

Complexity. The space complexity of the supervisor automata is determined by the number of automata and their number of transitions. To calculate the supervisor automata, the same abstraction steps as in the state representation approach are applied, which in the worst case can be $2n - 1$ steps. However, as the modular supervisor automata consist of distinguishers and supervisor components obtained after unsupervisability removal, two supervisor components may be produced at each step. This results in $2(2n - 1) = 4n - 2$ supervisor components in total. The space complexity to store each automaton is determined by its number of transitions. A deterministic automaton with $|Q|$ states and $|\Sigma|$ events has up to $|Q||\Sigma|$ transitions. Then the worst-case space complexity to store the supervisor components is $O((4n - 2)|Q||\Sigma|) = O(n|Q||\Sigma|)$, where $|Q|$ is the number of states of the largest automaton, and $|\Sigma|$ is the largest number of events after renaming. As in the case of the state representation-based supervisor, memory usage grows linearly in the number of automata and their states, however it also grows linearly in the number of events *after* renaming. In the worst case, there may be exponentially more events after renaming than before, although this case seems to be rare in practice.

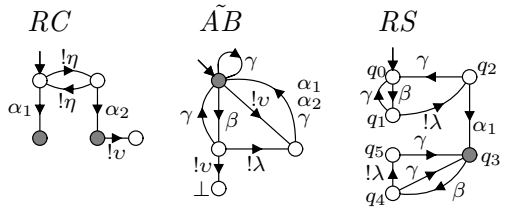

Figure 6: Automata-based supervisor.

State representation-based supervisors can solve the above problems of nondeterminism and not being least restrictive after transition removal, because all control decisions are based on the target states of the transitions in the original plant model $\mathcal{G}_0$. In addition, state representation-based supervisors in the worst-case are much smaller than automata-based supervisors.

7 Experimental Results

The state representation and automata-based compositional synthesis algorithms are implemented in the discrete event systems tool Supremica [1], and both implementations have been used to compute supervisors for several large discrete event system models. The test cases include complex industrial models and case studies from different application areas such as manufacturing systems and automotive body electronics, most of which have also been used as benchmarks in [21]. The following list gives an overview:

agv Automated guided vehicle coordination based on a Petri net model [23]. To make the example blocking in addition to uncontrollable, there is also a variant, **agvb**, with an additional zone at the input station.

aip Automated manufacturing system of the Atelier Inter-établissement de Productique [3].

fencaiwon09 Model of a production cell in a metal-processing plant [6].

fms2003 Large-scale flexible manufacturing system according to [34].

psl An assembly cell for toy cars, which are built up from seven parts [24]. **psl-big** is the basic model, **psl-restart** has additional transitions for restart, and **psl-partleft** has counters for extra parts.

tbed Model of a toy railroad system based on [13]. Three versions present different control objectives.

verriegel Models of the central locking system of a BMW car. There are two variants, a three-door model **verriegel3**, and a four-door model **verriegel4**. These models are derived from the KorSys project [26].

6link Models of a cluster tool for wafer processing previously studied for synthesis in [28].

The performance of the compositional algorithms is sensitive to the order in which automata are composed and simplified. The implementations underlying this working paper are based on a two-step selection procedure [8]. The first step computes a set of *candidates*, i.e., groups of automata to be considered for composition, and the second step selects a most promising candidate. The following heuristics [8, 31] are considered for the first step:

MustL considers for every event σ in the model the set of automata using this event σ as a possible candidate. This ensures that there is at least one local event, namely σ, after composition. As most of the abstractions depend on hiding of local events, this heuristic increases the possibility of abstraction.

Pairs considers as candidates all pairs of two automata that have at least one common event. This approach seeks to compose the system in small steps, only two automata at a time.

MinT considers as candidates all automata pairs containing the automaton with the fewest transitions. This heuristic tries to keep the intermediate results small.

MaxS considers as candidates all automata pairs containing the automaton with the most states. This typically results in all automata being composed together in a growing subsystem in a way similar to [28].

The second step of candidate selection attempts to identify the best candidate among the set of candidates from the first step. Again, a variety of alternatives [8, 31] are considered:

MaxL chooses the candidate with the highest proportion of local events. This heuristic attempts to increase the possibility of hiding and abstraction.

MinE chooses the candidate with the minimum number of events. This increases the possibility of observation-equivalence based abstraction.

MaxC chooses the candidate with the highest proportion of common events. This results in a smaller synchronous composition.

MinS chooses the candidate with the smallest estimated number of states after abstraction. The estimate is obtained by multiplying the product of the state numbers of the automata forming the candidate with the ratio of the number of events the candidate shares with other automata over the total number of events of the candidate.

Table 1: Experimental results.

Model				Automata			State Representation		
Name	**Cont**	**Nonb**	**Size**	**Time**	**Memory**	**Heuristic**	**Time**	**Memory**	**Heuristic**
agv	false	true	$2.6 \cdot 10^7$	0.17 s	19729	Pairs/MaxL	0.11 s	5585	MustL/MaxC
agvb	false	false	$2.3 \cdot 10^7$	0.30 s	27610	MaxS/MaxC	0.09 s	4305	MustL/MaxC
aip0alps	true	false	$3.0 \cdot 10^8$	0.04 s	360	Pairs/MinE	14.52 s	209	MustL/MaxC
fencaiwon09b	true	false	$8.9 \cdot 10^7$	0.05 s	5097	MustL/MinE	0.05 s	4185	MinT/MaxC
fencaiwon09s	false	false	$2.9 \cdot 10^8$	0.06 s	9626	MustL/MinSync	0.03 s	3615	MustL/MinE
fms2003	true	false	$1.7 \cdot 10^7$	26.10 s	265556	MinT/MinF	23.31 s	78675	MinT/MinE
psl_big	true	false	$3.9 \cdot 10^7$	0.05 s	1997	MustL/MinS	0.07 s	1721	Pairs/MaxC
psl_restart	true	false	$3.9 \cdot 10^7$	0.26 s	74240	MustL/MinE	31.19 s	37562	MustL/MaxC
psl_partleft	true	true	$7.7 \cdot 10^7$	42.07 s	603805	Pairs/MaxC	2.12 s	111313	Pairs/MinF
tbed_hisc1	true	false	$2.9 \cdot 10^{17}$	3.90 s	68431	Pairs/MinF	3.17 s	112361	Pairs/MinF
tbed_noderailb	true	false	$3.2 \cdot 10^{12}$	9.08 s	498	MaxS/MinS	4.54 s	1658	MinT/MinS
tbed_uncont	false	true	$3.6 \cdot 10^{12}$	9.06 s	131469	MaxS/MinS	3.48 s	132644	MustL/MinSync
verriegel3b	true	false	$1.3 \cdot 10^9$	0.42 s	34	MustL/MinSync	0.33 s	42	MustL/MinSync
verriegel4b	true	false	$6.2 \cdot 10^{10}$	0.63 s	34	MustL/MinSync	3.99 s	42	MinT/MinF
6linka	true	false	$2.4 \cdot 10^{14}$	0.53 s	66016	MinT/MinE	0.41 s	9045	Pairs/MaxC
6linki	true	false	$2.7 \cdot 10^{14}$	0.26 s	98280	MinT/MaxC	0.13 s	13002	MinT/MinE
6linkp	true	false	$4.2 \cdot 10^{14}$	0.55 s	91785	MinT/MaxL	0.41 s	8776	Pairs/MaxC
6linkre	true	false	$6.2 \cdot 10^{14}$	0.11 s	7153	Pairs/MinSync	0.10 s	2027	Pairs/MinSync

MinSync computes the synchronous composition of the automata in each candidate and chooses the candidate with the fewest states in the synchronous composition.

MinF chooses the candidate with the smallest number of other automata linked via events to the candidate's automata. This heuristic attempts to minimise event sharing between the candidate and the rest of the system.

In addition to the candidate selection strategy, the algorithms are controlled by state and transition limits. If the synchronous composition of a candidate exceeds 5,000 states or 1,000,000 transitions, that candidate is discarded and another is chosen instead.

Compositional synthesis has been attempted for all test cases, using both the state representation-based and automata-based algorithms, with all combinations of the candidate selection heuristics above. The results are shown in Table 1. For each test case, the table shows whether the model is controllable (Cont) or nonblocking (Nonb), and the number of reachable states of the uncontrolled system (Size). Next, the table shows for automata-based and state representation-based compositional synthesis, the total runtime (Time) and a memory estimate for the supervisor (Memory), in each case using the heuristic combination that produces the smallest supervisor. Supervisor size is used as the selecting criteria because

the memory usage plays an important role when it comes to implementing the supervisor in memory-limited devices like PLCs.

The memory for automata-based supervisors is estimated based on the numbers of states and transitions. Each state counts as one unit of memory, and each transition counts as two units of memory, estimating the amount of memory to form transition lists indexed by source states. For example, the supervisor automaton RS in Figure 6 has 6 states and 9 transitions, so its memory estimate is 24 units. All the supervisor automata in Figure 6 are estimated to use 59 units of memory together.

The memory for state representation-based supervisors is estimated based on the size of the maps. In Figure 5, there are two types of maps. Map μ_1 results from the abstraction of a single automaton, while maps μ_{23} and μ_{45} result from the synchronous composition of two automata. The abstraction map μ_1 can simply be stored as an array, mapping each of the five states of automaton C to a state in its abstraction $\tilde{C}$. The array contains only the sequence of states c'_0, c'_0, c'_1, c'_2, c'_3 of $\tilde{C}$, which are associated to the states c_i of C by their index i. This works because abstraction maps cover *all* states of the original automaton. Differently, synchronous composition maps only cover the *reachable* states of the synchronous composition. They are better stored as associative maps. A memory-efficient representation of map μ_{23} is the ordered sequence $\langle a_0, b_0, p_0, a_1, b_0, p_1, a_0, b_1, p_2 \rangle$. The abstracted state p_0 for the original state $a_0 b_0$, e.g., can be found using binary search [32]. Based on this, the array for map μ_1 stores 5 state names and uses 5 units of memory, while the associative maps for μ_{23} and μ_{45} use 9 and 18 units of memory respectively. This gives a total memory estimate of 32 units for all the supervisor maps in Figure 5, which is 27 units less compared to the automata-based approach.

All experiments have been run on a standard desktop PC using a single 3.3 GHz microprocessor and not more than 2 GB of RAM. Table 1 shows that both compositional synthesis algorithms successfully compute supervisors for all test cases in a few seconds or minutes. In most cases, the supervisor maps use less memory than automata-based supervisors as expected. Yet, there are few cases such as **tbed_hisc1** and **tbed_noderailb** where the automata-based supervisors use less memory. In these cases, closer analysis shows that the automata-based supervisor consists of only a few small automata resulting from unsupervisability removal, which have been abstracted substantially in the preceding steps, whereas the state representation-based supervisor includes maps from all intermediate steps, some of which are large.

The experiments also show that the runtimes and supervisor sizes are highly sensitive to the heuristics used. Every heuristic considered shows for at least one best result in Table 1. Fortunately, synthesis often completes in a matter of seconds, enabling the user to attempt several different heuristics or algorithms and choose the best result.

To examine the effect of transition removal, Figure 7 shows the total number of transitions encountered by the two algorithms with two fixed heuristic combinations. These results suggest that state representation-based synthesis, which supports transition removal, produces

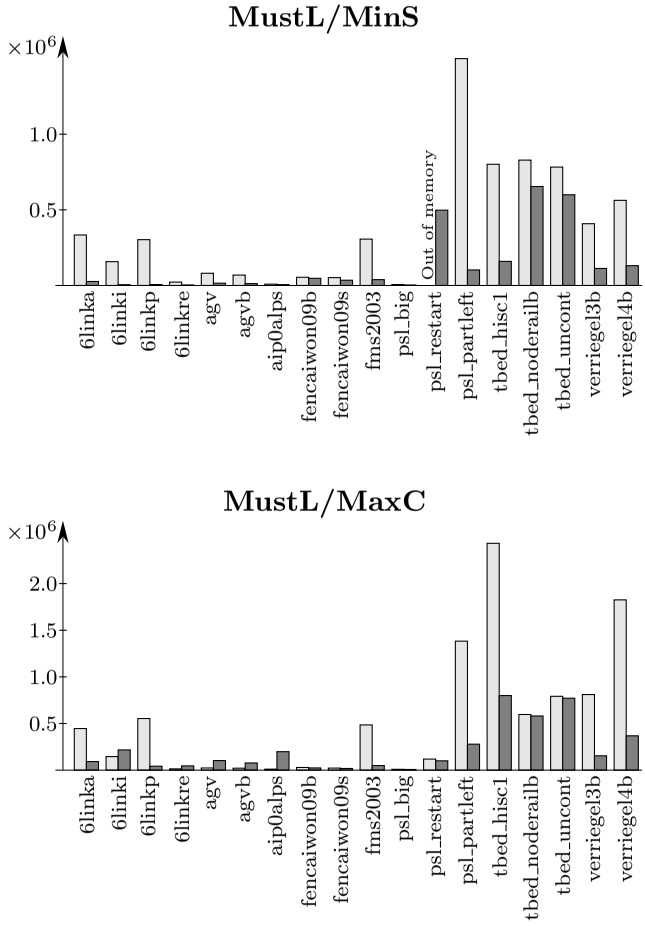

Figure 7: Number of transitions in automata-based synthesis (light) vs. state representation-based synthesis (dark).

significantly less transitions in general. Transition removal can even affect the success of the compositional approach. For example, the automata-based compositional algorithm fails to calculate a supervisor for **psl_restart** with the MustL/MinS heuristic combination. Yet, there are cases where state representation-based synthesis encounters more transitions, for example with the **agv** models under the MustL/MaxC heuristic combination. This can be explained by the fact that the MaxC heuristic is dependent on the number of events, which changes after renaming and in these cases seems to lead to poor decisions.

8 Conclusions

The working paper proposes a general framework for compositional synthesis of least restrictive, controllable, and nonblocking supervisors of large discrete event system models. The framework supports compositional reasoning using state merging, state removal, and transition removal abstractions that are guaranteed to preserve synthesis results. The final supervisor is a set of cascaded maps that represents the set of safe states. The algorithm has been implemented and its performance compared with the well-developed compositional synthesis algorithm [21] that returns a set of supervisor automata. The space complexity analysis and the experimental results suggest that supervisor maps in many cases require less memory than supervisor automata.

In future work, the authors would like to extend the approach and investigate the compositional synthesis of supervisors for finite-state machines augmented with bounded discrete variables.

References

[1] Åkesson, K., Fabian, M., Flordal, H., Malik, R.: Supremica—an integrated environment for verification, synthesis and simulation of discrete event systems. In: Proceedings of the 8th International Workshop on Discrete Event Systems, WODES'06, pp. 384–385. IEEE (2006)

[2] Åkesson, K., Flordal, H., Fabian, M.: Exploiting modularity for synthesis and verification of supervisors. In: Proceedings of the 15th IFAC World Congress on Automatic Control (2002)

[3] Brandin, B., Charbonnier, F.: The supervisory control of the automated manufacturing system of the AIP. In: Proceedings of Rensselaer's 4th International Conference on Computer Integrated Manufacturing and Automation Technology, pp. 319–324. IEEE Computer Society Press (1994)

[4] Brandin, B.A., Malik, R., Malik, P.: Incremental verification and synthesis of discrete-event systems guided by counter-examples. IEEE Transactions on Control Systems Technology **12**(3), 387–401 (2004). DOI 10.1109/TCST.2004.824795

[5] Bryant, R.E.: Symbolic Boolean manipulation with ordered binary-decision diagrams. ACM Computing Surveys **24**(3), 293–318 (1992). DOI 10.1145/136035.136043

[6] Feng, L., Cai, K., Wonham, W.M.: A structural approach to the non-blocking supervisory control of discrete-event systems. International Journal of Advanced Manufacturing Technology **41**, 1152–1168 (2009). DOI 10.1007/s00170-008-1555-9

[7] Feng, L., Wonham, W.M.: Supervisory control architecture for discrete-event systems. IEEE Transactions on Automatic Control **53**(6), 1449–1461 (2008). DOI 10.1109/TAC.2008.927679

[8] Flordal, H., Malik, R.: Compositional verification in supervisory control. SIAM Journal of Control and Optimization **48**(3), 1914–1938 (2009). DOI 10.1137/070695526

[9] Flordal, H., Malik, R., Fabian, M., Åkesson, K.: Compositional synthesis of maximally permissive supervisors using supervision equivalence. Discrete Event Dynamic Systems: Theory and Applications **17**(4), 475–504 (2007). DOI 10.1007/s10626-007-0018-z

[10] Graf, S., Steffen, B.: Compositional minimization of finite state systems. In: Proceedings of the 1990 Workshop on Computer-Aided Verification, *LNCS*, vol. 531, pp. 186–196. Springer (1990). DOI 10.1007/BFb0023732

[11] Harel, D.: Statecharts: a visual formalism for complex systems. Science of Computer Programming **8**(3), 231–274 (1987). DOI 10.1016/0167-6423(87)90035-9

[12] Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall (1985)

[13] Leduc, R.J.: PLC implementation of a DES supervisor for a manufacturing testbed: An implementation perspective. Master's thesis, Department of Electrical Engineering, University of Toronto, ON, Canada (1996). URL `http://www.cas.mcmaster.ca/~leduc`

[14] Luo, J., Nonami, K.: Approach for transforming linear constraints on Petri nets. IEEE Transactions on Automatic Control **56**(12), 2751–2765 (2011). DOI 10.1109/TAC.2011.2128590

[15] Malik, R., Flordal, H.: Yet another approach to compositional synthesis of discrete event systems. In: Proceedings of the 9th International Workshop on Discrete Event Systems, WODES'08, pp. 16–21. IEEE (2008). DOI 10.1109/WODES.2008.4605916

[16] Malik, R., Streader, D., Reeves, S.: Conflicts and fair testing. International Journal of Foundations of Computer Science **17**(4), 797–813 (2006). DOI 10.1142/S012905410600411X

[17] Miremadi, S., Åkesson, K., Lennartson, B.: Symbolic computation of reduced guards in supervisory control. IEEE Transactions on Automation Science and Engineering **8**(4), 754–764 (2011). DOI 10.1109/TASE.2011.2146249

[18] Mohajerani, S., Malik, R., Fabian, M.: Five abstraction rules to remove transitions while preserving compositional synthesis results. Working Paper 01/2012, Department of Computer Science, University of Waikato, Hamilton, New Zealand (2012). URL `http://hdl.handle.net/10289/6259`

[19] Mohajerani, S., Malik, R., Fabian, M.: Synthesis equivalence of triples. Working Paper 04/2012, Department of Computer Science, University of Waikato, Hamilton, New Zealand (2012). URL `http://hdl.handle.net/10289/7162`

[20] Mohajerani, S., Malik, R., Fabian, M.: Transition removal for compositional supervisor synthesis. In: Proceedings of the 8th International Conference on Automation Science and Engineering, CASE 2012, pp. 690–695 (2012). DOI 10.1109/CoASE.2012.6386447

[21] Mohajerani, S., Malik, R., Fabian, M.: A framework for compositional synthesis of modular nonblocking supervisors. IEEE Transactions on Automatic Control **59**(1), 150–162 (2014). DOI 10.1109/TAC.2013.2283109

[22] Mohajerani, S., Malik, R., Ware, S., Fabian, M.: On the use of observation equivalence in synthesis abstraction. In: Proceedings of the 3rd IFAC Workshop on Dependable Control of Discrete Systems, DCDS 2011, pp. 84–89 (2011). DOI 10.1109/DCDS.2011.5970323

[23] Moody, J.O., Antsaklis, P.J.: Supervisory Control of Discrete Event Systems Using Petri Nets. Kluwer Academic Publishers (1998)

[24] Parsaeian, S.: Implementation of a framework for restart after unforeseen errors in manufacturing systems. Master's thesis, Chalmers University of Technology, Göteborg, Sweden (2014)

[25] Ramadge, P.J.G., Wonham, W.M.: The control of discrete event systems. Proceedings of the IEEE **77**(1), 81–98 (1989). DOI 10.1109/5.21072

[26] KorSys Project: URL `http://www4.in.tum.de/proj/korsys/`

[27] Schmidt, K., Breindl, C.: Maximally permissive hierarchical control of decentralized discrete event systems. IEEE Transactions on Automatic Control **56**(4), 723–737 (2011). DOI 10.1109/TAC.2010.2067250

[28] Su, R., van Schuppen, J.H., Rooda, J.E.: Aggregative synthesis of distributed supervisors based on automaton abstraction. IEEE Transactions on Automatic Control **55**(7), 1267–1640 (2010). DOI 10.1109/TAC.2010.2042342

[29] Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pacific Journal of Mathematics **5**(2), 285–309 (1955)

[30] Vahidi, A.: Efficient analysis of discrete event systems—supervisor synthesis with binary decision diagrams. Ph.D. thesis, Chalmers University of Technology, Göteborg, Sweden (2004)

[31] Ware, S., Malik, R., Mohajerani, S., Fabian, M.: Certainly unsupervisable states. In: Proceedings of the 2nd International Workshop on Formal Techniques for Safety-Critical Systems, FTSCS 2013, pp. 3–18 (2013)

[32] Wirth, N.: Algorithms and Data Structures. Prentice-Hall (1986)

[33] Wong, K.C., Wonham, W.M.: Modular control and coordination of discrete-event systems. Discrete Event Dynamic Systems: Theory and Applications **8**(3), 247–297 (1998). DOI 10.1023/A:1008210519960

[34] Zhou, M.C., Dicesare, F., Rudolph, D.L.: Design and implementation of a Petri net based supervisor for a flexible manufacturing system. Automatica **28**, 1199–1208 (1992). DOI 10.1016/0005-1098(92)90061-J